UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

USO Y VENTAJAS DE ELASTICSEARCH EN BASES DE DATOS NO RELACIONALES

Carla Civantos Martos
Tutora: Ana María González Marcos

JUNIO 2019

USO Y VENTAJAS DE ELASTICSEARCH EN BASES DE DATOS NO RELACIONALES

AUTORA: Carla Civantos Martos
TUTORA: Ana María González Marcos

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen

El origen de este Trabajo Fin de Grado (TFG) se basa en la inquietud de conocer qué son las bases de datos no sólo SQL (Structured Query Language) (NoSQL) y cómo interactúan éstas con potentes motores de búsqueda como la herramienta Elasticsearch.

Por lo tanto, el TFG se centrará en el motor de búsqueda Elasticsearch, plataforma implementada en código abierto y creada sobre Lucene que es una biblioteca con funciones destinadas a la recuperación de información que Elasticsearch utiliza para generar su interfaz.

Para poder realizar una búsqueda eficiente, previamente es necesario estructurar correctamente la recopilación de la información para su posterior procesamiento y análisis final. En el desarrollo del TFG se mostrará cuál es la secuencia de pasos a realizar bajo el dominio de la API ELK. En la memoria se definen y explican el conjunto de comandos, funciones y protocolos que forman la API ELK implementada para permitir a los desarrolladores crear programas específicos para las necesidades generadas en entornos que manejan cantidades altas de información y que deben ser procesadas casi en tiempo real.

ELK es un conjunto de tres herramientas especializadas: Elasticsearch, Logstash y Kibana. Cada una de estas herramientas puede ser utilizada independientemente, pero la unión de todas ellas hace una combinación idónea para la gestión de grandes cantidades de datos permitiendo detectar incidencias en tiempo real, optimizar el tiempo de respuesta y ofrecer soluciones de escalabilidad de sistemas.

En la memoria del TFG se desarrolla cómo interactúan entre sí cada una de las herramientas. A modo de resumen, se inicia el proceso con el motor de procesamiento y recopilación de los datos, Logstash. En segundo lugar, llega el turno de Elasticsearch encargado de indexar los datos que le llegan desde el programa previo (Logstash), su función principal es el almacenamiento de los datos, pero es el paso clave para realizar las búsquedas en el servicio; recibe las consultas de los clientes y las ejecuta buscando en sus componentes de almacenaje y devolviendo los resultados esperados.

Finalmente, aparece Kibana, el último programa del proceso, su función es la visualización y análisis de datos, una vez que el motor de búsqueda devuelve los resultados de las consultas, estos pueden ser monitorizados en diferentes formatos y, además, ofrece unas funciones de filtros para personalizar los resultados obtenidos.

Analizando el auge que ha alcanzado la plataforma ELK, mostraremos casos de uso en empresas como Telefónica, Orange, Ebay, Wikipedia, etc. Las razones para incorporar la plataforma ELK en el día a día de las empresas se deben, principalmente, a su flexibilidad, rapidez en las búsquedas y personalización de las bases de datos, además del alto rendimiento que aporta al servicio.

Palabras clave

BBDD, NoSQL, Elasticsearch, motor de búsqueda, Beats, Logstash, Kibana.

Abstract

The object of this Final Degree Project (TFG) is based on the knowledge of not only SQL databases (NoSQL) and how they work with search engines such as the Elasticsearch tool.

Therefore, the project will focus on the search engine Elasticsearch, which is a platform implemented in open source and created on Lucene, which is a library with the functionality of retrieving information that Elasticsearch uses to develop its interface.

In order to do an efficient search, the first step is to structure the information to process and analyse it. During the memory, all the steps of the ELK user interface will be shown to know the commands, functions and protocols that allow developers to create specific programs to handle large amounts of information that can be processed in real time.

ELK needs three tools to work properly: Elasticsearch, Logstash and Kibana. Each one of these tools can be used independently, but the union of all of them makes it possible to manage large amounts of data that can detect incidents in real time, optimize response time and provide system scalability solutions.

In the TFG memory, each of the tools works with each other. In summary, the process begins with the data processing and collection engine, Logstash. Secondly, Elasticsearch indexes the data that comes from the previous tool (Logstash), its main function is the storage of the data, but it is the main step to perform the searches in the service. Also, it receives the queries form the clients and executes them searching in its storage components and returning the expected results.

Finally, it´s the turn of the last tool of the process and this is Kibana. Its function is the visualization and analysis of data. When Elasticsearch returns the results of the queries, these can be monitored in different formats. Furthermore, it offers some functions of filters to customize the results.

Analysing the growth that the ELK platform has achieved, will be shown cases of use in companies such as Telefónica, Orange, eBay, Wikipedia, etc.

The reasons why the ELK platform is used today in companies are due to the flexibility, the search speed and the customization of the databases, also due to the high performance it brings to the service.

Keywords

BBDD, NoSQL, Elasticsearch, Search Engine, Beats, Logstash, Kibana.

Agradecimientos

A mi familia por la confianza, el cariño, el apoyo y las facilidades que siempre me han dado.

A Sergio por confiar en mí y darme los ánimos y empujones necesarios para seguir adelante.

A mi tutora, Ana, que sin ella esto no habría sido posible, gracias por la oportunidad.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

Glosario

ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
APM	Application Performance Monitoring
BBDD	Bases de datos
BLOB	Binary Large Objects
BSON	Binary JSON
CPU	Central Processing Unit
CQL	Cassandra Query Language
DHT	Distributed Hash Table
ELK	Elasticsearch, Logstash y Kibana
E/S	Entrada / Salida
GQL	Graph Query Language
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
NoSQL	Bases de datos no relacionales
NRT	Near Real-Time or Nearly Real-Time
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
XML	Extensible Markup Language

1 Introducción

1.1 Motivación

Actualmente la información es un recurso disponible para todo el mundo, esto conlleva avances, grandes mejoras en la sociedad y la investigación, pero a su vez ha supuesto una carga y un desafío para las grandes compañías debido a la cantidad de información procesada a diario por usuarios, la necesidad de realizar un tratamiento de datos, unido al alto crecimiento de la Web junto con las nuevas tecnologías, por ejemplo ahora nos enfrentamos a un nuevo desafío con la liberación de la tecnología 5G.

El constante crecimiento de las aplicaciones supone un aumento de la información y a su vez, procesar grandes volúmenes de datos en tiempo real para la satisfacción de los usuarios. Estas necesidades son las que llevaron a las compañías a priorizar el rendimiento y la rápida respuesta frente a la consistencia de los datos, creando un auge en el uso de las bases de datos no relacionales en general y bases de datos NoSQL (bases de datos no sólo SQL) como caso particular. En la memoria nos centraremos en bases de datos NoSQL.

Las bases de datos no relacionales han sido diseñadas con la intención de crear aplicaciones nuevas, como se puede observar en la Figura 1-1, la diferencia de estructuración; las NoSQL no siguen una estructuración específica si no que pueden configurarse y almacenar una gran variedad de modelos de datos como documentos, gráficos, clave-valor, etc., solucionando problemas de accesibilidad, solucionando situaciones de rendimiento y escalabilidad que las BBDD (bases de datos) relacionales no son capaces de solventar y soportando una gran cantidad de datos gracias a su estructura distribuida con el uso de tablas Hash en algunos casos.

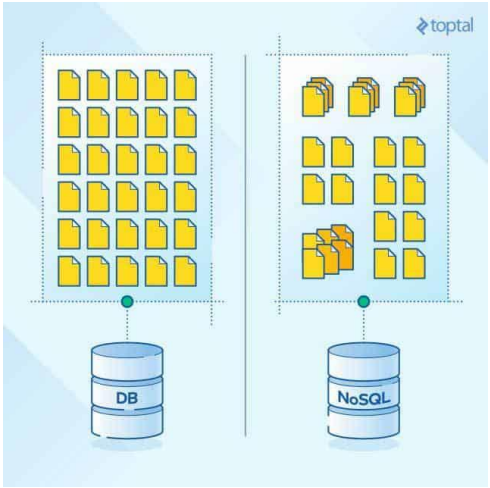

Figura 1-1: Diferencia entre la estructuración de BBDD relacionales y NoSQL (1)

El objetivo de este modelo de BBDD es conseguir la manipulación de una gran cantidad de datos. Sus principales características son:

- Flexibilidad, una de las principales características es la variedad de modelos de datos que pueden procesar las BBDD NoSQL, no tienen que seguir un esquema ni una estructura fija para almacenar los datos por lo que facilita la personalización de las BBDD y aplicaciones procesando únicamente los datos imprescindibles. También da opción a almacenar estructuras complejas en un solo documento, aumentando la información, pero sin dificultar su proceso.

- Escalabilidad horizontal, en consecuencia, a la importancia del rendimiento del sistema, estas BBDD trabajan con clústeres distribuidos que pueden aumentar físicamente, asegurando que el sistema va a conocer la disponibilidad de todos los clústeres y sus nodos.

- Altamente funcional, los datos de estas bases siguen unos requisitos específicos para sus modelos de datos, esto facilita la funcionalidad de cualquier API (Application Programming Interface) debido a que la vuelve más personal.

- Consistencia eventual, con motivo de buscar un mayor rendimiento, las BBDD NoSQL actualizan los cambios de los datos con el tiempo, no ofreciendo los últimos datos disponibles, si no, unos que pueden ser obsoletos. Esta es una de las grandes diferencias con las BBDD relacionales que se basan en las características ACID (Atomicidad, Consistencia, "Isolation" Aislamiento y Durabilidad) ofreciendo una restricción en la integridad de los datos y reglas definidas.

Existen varios tipos de bases de datos no relacionales, como se ha indicado anteriormente, esto depende de cómo se requiere almacenar la información:

- BBDD clave-valor, es uno de los modelos más populares y con funcionalidad sencilla. Cada elemento se identifica con una clave única y utiliza tabla hash para el almacenamiento y la búsqueda de los elementos. La clave es transformada en un número que identifica la posición en la tabla donde se localiza el valor, esto aporta rapidez y almacena la información en forma de cadena, JSON (JavaScript Object Notation) o BLOB (Binary Large Objects).
  La Figura 1-2 muestra un ejemplo de almacenamiento de este tipo de BBDD.

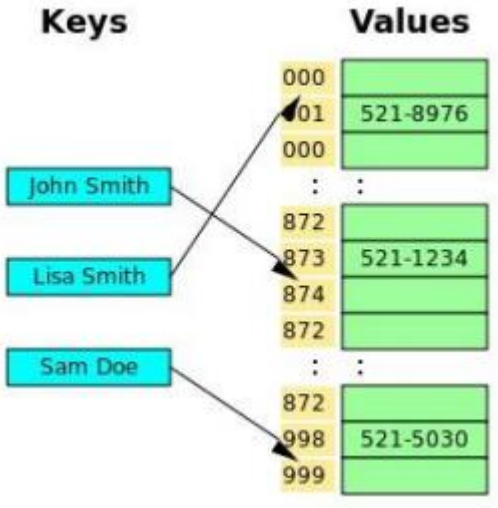

Figura 1-2: Almacenamiento en BBDD clave-valor (2)

- BBDD documentales, donde la palabra lo indica, los tipos de datos que soporta son documentos, normalmente con estructuras como JSON, BSON (Binary JSON) o XML (Extensible Markup Language) y una clave única para cada registro, también permite búsquedas por clave-valor y realizar consultas avanzadas. Son las más versátiles y su uso se puede enfocar en casos donde también encajarían las bases de datos relacionales. Este tipo de bases almacenan documentos con toda la información, pero a su vez existen etiquetas donde se pueden consultar partes de la información del documento, los datos más relevantes son los que se clasifican como etiquetas, en la Figura 1-3 se puede ver un ejemplo y además se puede ver que encajaría en una BBDD relacional.

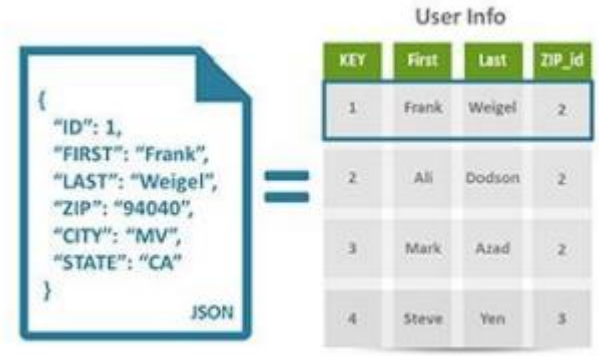

Figura 1-3: Almacenamiento en BBDD de documentos (2)

- BBDD en grafos, como se puede observar en la Figura 1-4, la información se representa como nodos de un grafo y sus relaciones son las aristas. Para poder recorrer estas BBDD se hace uso de la teoría de grafos; Este esquema de BBDD están enfocadas al uso de datos conectados como por ejemplo las redes sociales, motores de recomendaciones, detección de fraude, etc. Esta BBDD sería la más parecida a las BBDD relacionales en cuanto a la estructuración que sigue.

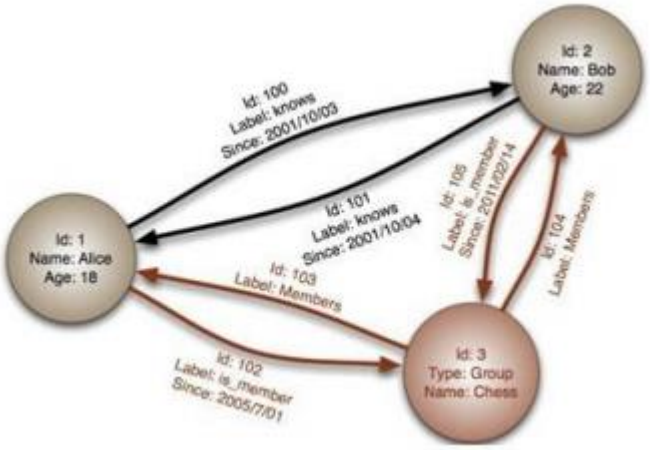

Figura 1-4: Almacenamiento en BBDD de grafos (2)

- Existen otras BBDD no relacionales como son, orientadas a objetos y en memoria, este último tipo son muy útiles para la solución de problemas y visualizar el estado de aplicaciones o análisis de datos en tiempo real.

La variedad de tipos de bases NoSQL se debe a las características de cada aplicación. Dependiendo del diseño de la aplicación a desarrollar, se pueden crear nuevos tipos de BBDD o bien utilizar alguno de los nombrados anteriormente.

El nacimiento de las BBDD NoSQL no busca sustituir a las relacionales, si no, ayudar, dependiendo de las necesidades que tenga la aplicación creada, utilizando las BBDD que mejor se complemente al diseño de la aplicación. Algunas particularidades son el volumen de datos y su crecimiento, los costes y la técnica de la escalabilidad, el estudio de uso del sistema por los usuarios y una de las más importantes, los campos distintos en la información almacenada.

Para tener una idea a la hora de crear una aplicación nueva, en la Tabla 1-1 se han recopilado las propiedades con mayor relevancia.

Tabla 1-1: Propiedades de las BBDD

Propiedades / Características	BBDD Relacionales	BBDD NoSQL
Arquitectura distribuida	✗	✓
Consistencia	✓	✗
Escalabilidad horizontal	✗	✓
Esquema entidad-relación	✓	✗
Evitar cuellos de botella	✗	✓
Fiabilidad	✓	✗
Lenguaje SQL	✓	✗
Máquinas con pocos recursos	✗	✓
Manejo grandes cantidad de datos	✗	✓
Optimización de almacenamiento	✓	✗
Optimización de datos	✗	✓
Reduce redundancia	✓	✗
Rendimiento	✗	✓
Sistema complejo	✗	✓
Tolerancia a fallos	✗	✓

Las plataformas de BBDD no relacionales más conocidas en el mercado actual son:

- Almacenando datos binarios, Cassandra (3) con el lenguaje CQL (Cassandra Query Language) o BigTable (4) con el lenguaje GQL (Graph Query Language).
- Almacenando datos como documentos, MongoDB con documentos de tipo JSON.
- Almacenando datos estructurados como grafos, Neo4j (5) e InfoGrid (6).

Existen grandes empresas que utilizan bases de datos no relacionales debido a la carga de datos, al crecimiento exponencial de estos y la interacción de los usuarios con la aplicación, algunas de estas compañías son: Facebook, Twitter, Yahoo, Instagram, Github, Infojobs, Youtube, etc.

1.2 Objetivos

El principal objetivo de este trabajo es comprender Elasticsearch como motor de búsqueda, conocer su nacimiento, la necesidad de su uso e investigar su arquitectura, algoritmos, usos prácticos y el resto de las herramientas que son necesarios para su correcto funcionamiento.

En primer lugar, algunas de las preguntas más relevantes son, ¿de dónde saca toda la información?, ¿cómo almacena dicha información? y ¿cómo la procesa?; a estas tres preguntas se les puede asociar la funcionalidad de cada programa fundamental para el inicio del conocimiento de este motor de búsqueda.

Elasticsearch aumenta su potencial cuando se trabaja en cooperación con las herramientas Logstash y Kibana. La API ELK (Elasticsearch, Logstash, Kibana) incluye las tres herramientas (Elasticsearch, Logstash y Kibana) para gestionar registros, permitiendo a los usuarios de ELK recoger, organizar y preparar datos con fines analíticos (pudiendo ser el análisis en tiempo real) y desde distintos servidores.

En primer lugar, nos encontramos con la herramienta Logstash, encargada de agrupar y procesar los datos obtenidos de los servidores, transformarlos y enviarlos a la BBDD de Elasticsearch.

Una vez almacenada e indexada la información, llega el turno de Elasticsearch como motor de búsqueda. Su principal objetivo es buscar la información solicitada a través de conjuntos de claves y valores. Más adelante se explicará su funcionamiento y el algoritmo de búsqueda que necesita para desarrollar su función.

Finalmente, se consideró necesaria una interfaz gráfica donde el usuario pueda ver todo el proceso y realizar las búsquedas de los datos. Para este proceso, se creó Kibana como una plataforma de agrupación y visualización.

1.3 Organización de la memoria

Este trabajo consta de un resumen donde se explica brevemente cuáles son las inquietudes que nos han llevado a investigar Elasticsearch y se presenta una visión general de lo que se abordará en el documento.

El primer capítulo se corresponde con la introducción donde se explica qué son las BBDD no relacionales, la razón de su nacimiento y las diferencias que se encuentran entre éstas y las BBDD relacionales, también se tratan los objetivos que se quieren conseguir con este trabajo, que son la explicación y entendimiento del motor de búsqueda Elasticsearch.

A continuación, el desarrollo de los capítulos de la memoria se puede seguir a través de la Figura 1-5, donde se muestra en forma de esquema, el conjunto de herramientas que forman ELK. La memoria se irá desarrollando sobre estas herramientas de izquierda a derecha, desde la base más lejana al usuario final, hasta la interfaz gráfica con el que interactúa el usuario.
El capítulo dos se centra en la herramienta Logstash, la parte izquierda de la Figura 1-5. Se definirá qué son los contenedores Beats[1], se realiza una introducción a Logstash, definiéndolos y comentando sus funcionalidades. También se definen algunos conceptos importantes para poder entender su arquitectura y su funcionamiento. La arquitectura interna del programa se guía a través de otro esquema en dicha sección y se explica su funcionamiento siguiendo el esquema como guía. Además, se incluye un apartado donde se considera una alternativa al programa Logstash, en él se explican diferencias y razones por las que utilizar una opción u otra.

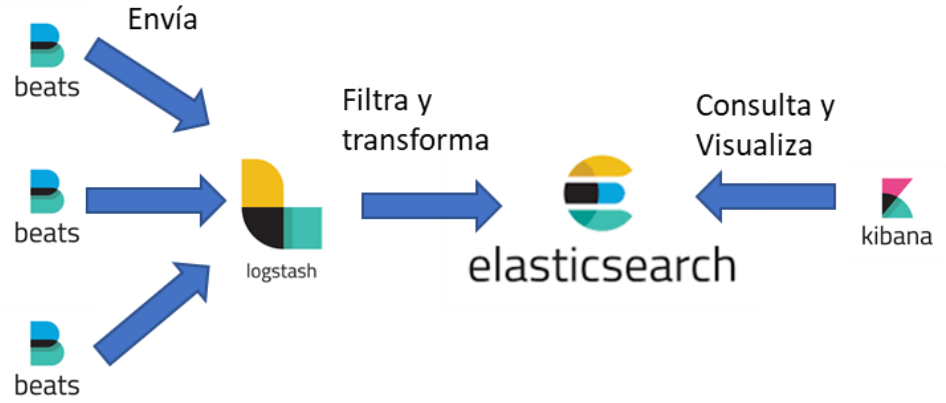

Figura 1-5: Esquema de la arquitectura de ELK (7)

El capítulo tres se centra en la herramienta Elasticsearch como motor de búsqueda, parte central de la Figura 1-5. En este capítulo, se explicará su funcionalidad y se darán conceptos básicos para comprender su arquitectura y funcionamiento interno. Una vez adquiridos los conceptos, se explica brevemente los pasos a seguir para configurar Elasticsearch, estas configuraciones se centran en la creación de los componentes que forman la arquitectura interna de la aplicación, que seguidamente, será desarrollada haciendo hincapié en las funciones de mapeo y análisis de datos, muy importantes en todo el procedimiento. Por último, se explica el funcionamiento de la plataforma desde que los datos salen de Logstash y se indexan en Elasticsearch.

[1] Beats son cargadores de datos que se instalan en servidores y funcionan como agentes que envían información operacional a Elasticsearch.

El capítulo cuatro de la memoria se centra en el programa Kibana, un visualizador y analizador de datos. Este capítulo enumera de forma breve los tipos de visualizadores que contiene el conjunto y muestra algunos ejemplos, los más conocidos.

Para finalizar, se muestra un pequeño listado con las empresas de magnitud que utilizan este conjunto de herramientas para mejorar sus servicios. Este capítulo busca concienciar de la importancia de la plataforma ELK en la actualidad.

Seguidamente la memoria finaliza con una conclusión sobre Elasticsearch y sus beneficios.

2 Logstash

2.1 Definición, conceptos y terminología

Como se ha introducido en el apartado Objetivos del capítulo anterior, la herramienta Logstash es una de las partes importantes para el motor de búsqueda Elasticsearch. Logstash es el encargado de recolectar, almacenar y procesar por primera vez los datos en cualquier formato y tamaño que se almacenarán posteriormente en las BBDD.

Logstash es el motor central de flujo, recopila y procesa los datos, se organiza en varios *pipelines* para poder mezclar, combinar y organizar las entradas, filtros y salidas. Dentro de esta plataforma, existen otros módulos y conceptos importantes para el proceso de parseo. Esto quiere decir, para enriquecer los datos y conocerlos mejor, se realiza la limpieza y transformación de los registros durante la entrada en tiempo real, en el índice o en la salida. Estas funciones se llevan a cabo con agregaciones, comparación de patrones, mapeo y capacidades de búsqueda dinámica.

A continuación, describiremos los pasos del *pipeline*. En primer lugar, nos encontramos con los inputs plugins, encargados de permitir que Logstash pueda leer unos datos específicos.
Elastic soporta plugins[2] de entradas diferentes, entre otros, los más usados o conocidos son: beats, estos reciben archivos de distintos Elastic Beats framework; elasticksearch, lee resultados de las consultas de otro clúster de Elasticsearch; exec, capta las salidas de comandos de Shell; file, transmite datos desde ficheros; github, transforma las solicitudes http en eventos, google o twitter.

Los anteriormente citados beats, son cargadores de datos ligeros, esto quiere decir que la plataforma llamada Beats se encarga de mandar los datos desde otras máquinas y sistemas a Logstash o Elastisearch. Son programas que se asientan en los servidores desde los que se mandan los datos al motor de búsqueda, mediante contenedores o en forma de funciones, estos datos son recopilados y enviados, en nuestro caso, a la plataforma Logstash para transformarlos y analizarlos.

Dependiendo del tipo de dato que almacenen los servidores, se pueden asentar diferentes programas Beats para mejorar el proceso de envío, estos tipos son: Filebeat (8), para archivos de registro o logs, su cliente es liviano y fácil de usar; Metricbeat (9), Packetbeat (10), Winlogbeat (11), Auditbeat (12), Heartbeat (13), Functionbeat (14).

La aparición de tuberías en Logstash se debe a una serie de problemas que surgieron al cargar y procesar datos con distintos formatos. Uno de los problemas principales era la necesidad de múltiples flujos para un aumento del rendimiento de Logstash. La primera solución a este problema se realizó eliminando condicionales: los eventos se etiquetaban al inicio y se creaban ramas condicionales en los filtros y salidas. Esto suponía problemas de administración con el aumento de la complejidad, falta de aislamiento por congestión.

[2] Plugin es una aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software**.** Podría decirse que se trata de un complemento.

Por estas razones se necesitaron múltiples instancias y la comunicación se realizó a través de tuberías. Una tubería tiene varios de los tres principales elementos que son: entrada, filtro y salida. Los complementos de entrada cogen datos de una fuente, los de filtro modifican los datos según sea necesario y los de salida escriben los datos en un destino. Podemos ver la estructura descrita en la imagen Figura 2-1, donde queda indicar que cuando se habla de canalización se refiere al proceso de creación de tuberías protagonistas durante la funcionalidad de Logstash.

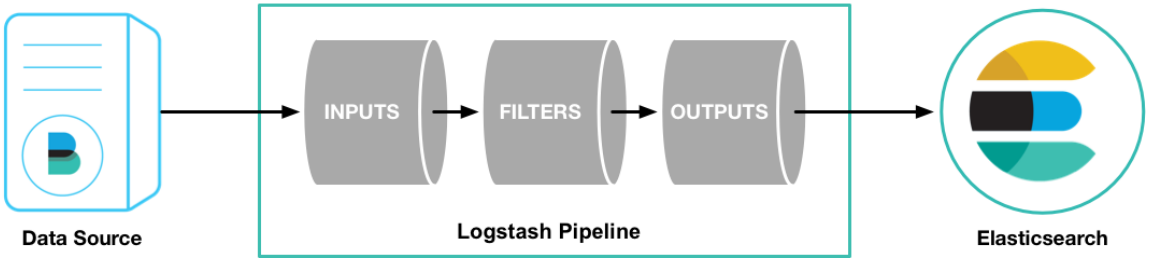

Figura 2-1: Estructura canalización básica de Logstash (7)

Los filtros son los encargados de leer y transformar la información según se necesite.
En el caso de funcionar con varios *pipelines*, se pueden configurar por separado con filtros separados, generando beneficios en el rendimiento y utilizando los plugins de salida de manera eficiente. A continuación, se enumeran los filtros más comunes:

- Grok: se utiliza para estructurar datos no estructurados para facilitar que estos puedan ser buscados.
- Mutar: puede cambiar el nombre, eliminar, reemplazar y modificar campos en sus eventos.
- Drop: envía eventos de depuración.
- Clonar: hace una copia de otro evento, insertando o sustrayendo campos.
- Geoip: agrega información sobre la dirección IP, marca de tiempo, ubicación geográfica.

Otro concepto importante es el **evento**, unidad de información que contiene marca de tiempo y datos. Los eventos llegan por una entrada, son analizados y marcados durante el paso por las tuberías. Los eventos están formados por campos, también conocidos como propiedades.

2.2 Arquitectura interna

Para comprender la arquitectura de esta herramienta, vamos a seguir una explicación paso por paso de su completa funcionalidad.

En primer lugar, los beats son los complementos que inician este proceso, estos envían datos desde una fuente o máquina, que puede ser distinta o la misma que ejecuta Logstash estos beats ya enumerados anteriormente, son configurados para recopilar y enviar al complemento “beats de entrada”, encargado de recibir los eventos o datos del marco de Beats, para su procesamiento. Se configuran indicando la ruta de entrada, el tipo de dato de entrada y el puerto de salida.

A continuación, se crea una tubería de configuración de Logstash que utiliza el complemento “beats de entrada” para recibir los datos. En la configuración se especifica el puerto de entrada que debe coincidir con el de salida del complemento beat utilizado en el primer paso y el formato estándar de salida.

El archivo de configuración de la tubería se utiliza para ejecutar varias tuberías en una misma instancia del programa. Se pueden crear tantos archivos de configuración como canales que le manden información se necesiten para trabajar con distintos tipos o formatos de datos que recibe Logstash.

En el siguiente paso, llega el turno de los plugins de filtro, su función es, una vez leído los datos, estos deben ser analizados para cambiar su formato a uno con datos específicos y obtener información adicional de los registros.

Para este primer filtro se utiliza el complemento predeterminado grok; este complemento permite estructurar los datos para que puedan ser mejor buscados, primero se deben buscar patrones de interés para su posterior uso, una vez se conozcan estos patrones, se deben especificar en el archivo de tubería de configuración y finalmente cuando los datos hayan pasado por el filtro, se obtienen los datos con representación JSON.

Terminado el proceso de filtrado, ha llegado el momento de indexar los datos en un clúster de Elasticsearch, esto debe ser especificado, con el puerto http que conecta con Elasticsearch, como en los casos anteriores en el archivo de tubería de configuración.

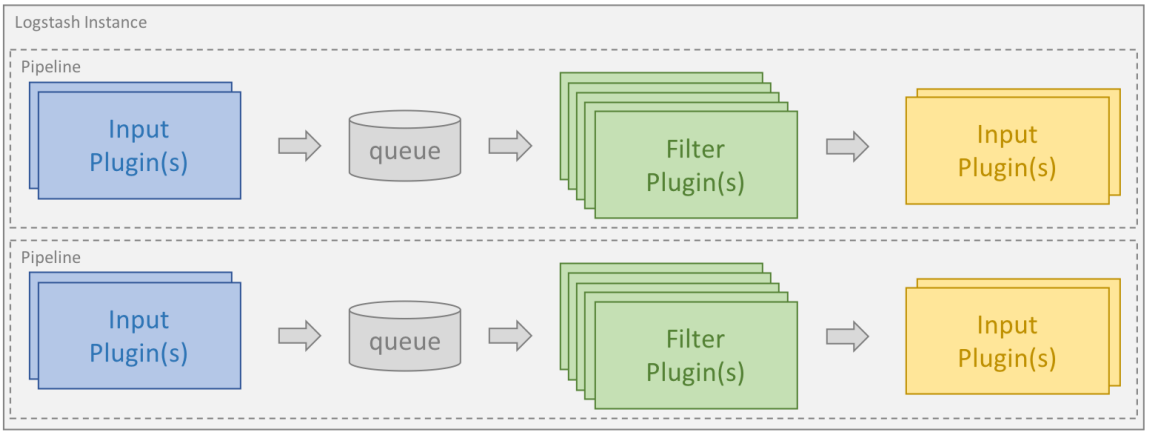

Figura 2-2: Arquitectura interna de Logstash (7)

Como se puede observar en la Figura 2-2, además de los componentes nombrados durante el desarrollo del funcionamiento de Logstash, también existen colas limitadas en memoria o disco para almacenar los eventos en varios puntos del proceso. Se pueden encontrar colas entre las etapas de la tubería de entrada, entre los filtros y previas a la salida. La información almacenada en estas colas se puede perder si el programa termina de forma inesperada o insegura, pero esto se puede solucionar con las colas persistentes, almacenan los eventos en una cola interna en el disco, o las colas de mensajes no entregados, almacenan eventos que no se han podido procesar en disco.

2.3 Otra alternativa: Nodos de ingestión

Logstash ha ido mejorando y aumentando componentes para mejorar en su funcionalidad. Pero existe una forma alternativa de procesar documentos en Elasticsearch antes de la indexación de los datos, los nodos de ingestión. Estos nodos soportan arquitecturas más sencillas con menor cantidad de componentes y la información es enviada, procesada e indexada directamente en Elastic.

Una de las diferencias principales entre Logstash y los nodos de ingestión es la forma de entrar y salir los datos. Por un lado, los nodos de ingestión insertan datos mediante solicitudes masivas y como se puede ver en la Figura 2-3 los nodos de ingestión se ejecutan dentro del flujo de indexación de Elasticsearch, debe existir un proceso que escriba los datos continuamente. Logstash al ser una herramienta separada de Elasticsearch, puede enviar datos sin afectar el funcionamiento de Elastic.

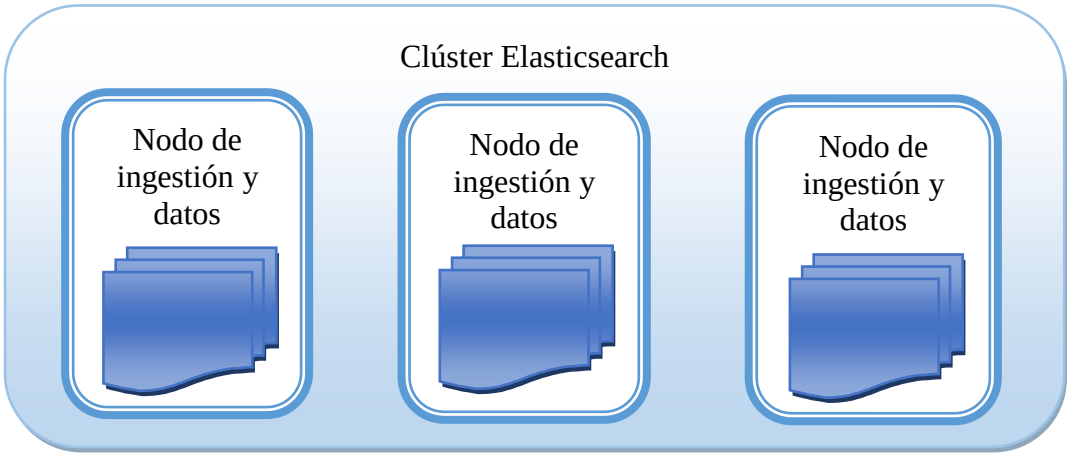

Figura 2-3: Esquema de la estructura de un clúster donde los nodos de datos son a su vez de ingestión

Además, los nodos de ingestión no pueden extraer datos de fuentes externas. Sin embargo, Logstash puede extraer datos de fuentes externas y aceptar datos insertados por los clientes por TCP (Transmission Control Protocol), UDP (User Datagram Protocol) y HTTP (HyperText Transfer Protocol) debido a la variedad de complementos de entrada y salida que tiene.

Por otro lado, en el caso de que Elastic no pueda almacenar más datos en sus nodos de datos, los nodos de ingestión tampoco podrán aceptar datos, mientras en el caso de Logstash puede seguir almacenando datos gracias a su mecanismo de colas y a su buffer a nivel local. Esta diferencia es muy importante porque las herramientas que escriben datos directamente en Elasticsearch perderán datos y Logstash garantiza al menos una entrega de datos y almacenaje durante los picos de ingestión.

Aunque existen bastantes diferencias entre las dos alternativas, estas pueden usarse en conjunto, mandar algunos a Logstash y otros a los nodos, usando el que tenga más sentido para cada flujo de datos y esto puede facilitar el mantenimiento de la arquitectura.

Viendo las ventajas y desventajas de cada opción, lo óptimo sería seleccionar la que mejor se ajuste a los criterios.

3 Elasticsearch

3.1 Definición, conceptos y terminología

Elasticsearch es una herramienta popular hoy en día debido a la gran cantidad de datos que se generan en la actualidad y a un ritmo constante. “Big Data” es el término que se ha acuñado para referirse al gran volumen de datos, tanto estructurados como no estructurados, que generan los negocios cada día.

La necesidad de Elasticsearch surgió para realizar búsquedas efectivas. El primer lanzamiento de Elasticsearch aparece en 2010. Se trata de un servidor de búsquedas basado en Apache Lucene (15), Lucene es una biblioteca de recuperación de datos que aporta las funcionalidades de indexación y búsqueda que Elasticsearch implementa, además está desarrollado en código abierto escrito en Java. Con las cantidades de datos que se generan a diario, navegar entre ellos era una actividad complicada, por ejemplo, para buscar patrones entre tanta información, hacerla legible y ordenarla casi en tiempo real, gracias a su estructuración de los datos mediante índices invertidos. Tiempo real o NRT (Near Real-Time), significa que existe una pequeña latencia en Elastic desde que se indexa un documento hasta que éste puede ser buscado.

Como se ha indicado en la introducción, Elasticsearch es un motor distribuido de búsqueda y análisis, se utiliza en aplicaciones con características de búsquedas complejas. Algunos conceptos que se deben tener claros serán desarrollados en este apartado.

Para comprender su arquitectura interna, primero vamos a definir algunos conceptos importantes de este motor de búsqueda.

3.1.1 Clúster

Elasticsearch está formado por clústeres, esto es una colección de uno o más nodos que almacenan los datos y aportan capacidades de búsqueda e indexación en todos los nodos. Una característica importante de este componente es que todos los clústeres deben tener un nombre único para que los nodos no se unan a los incorrectos. Cuando se unen o salen los nodos de un clúster, este se reorganiza automáticamente para distribuir uniformemente los datos entre los nodos disponibles. Este proceso se realiza con ayuda del nodo maestro que es elegido por el grupo y puede ser reemplazado.

3.1.2 Nodo

Dentro del clúster se encuentran los nodos. Un nodo es un servidor único, este almacena los datos y participa en las búsquedas de indexación del clúster. Como estos últimos, los nodos deben tener un nombre propio único asignado desde el inicio y llamado UUID (Universally Unique Identifier). Dentro de un clúster, los nodos pueden comunicarse mediante TCP.

Todos los nodos conocen todos los demás nodos del clúster y se reenvían entre ellos las solicitudes de los clientes hasta llegar al indicado. Existen varios tipos de nodos:

- Nodo elegible maestro, es el responsable de crear o eliminar un índice, hacer seguimiento de los nodos que forman parte del clúster y decidir qué fragmentos asignar a qué nodos.
- Nodo de datos, contienen los fragmentos con los documentos que ha indexado.
- Nodo de ingesta, ejecutan tuberías de subprocesos para mejorar la administración del consumo de memoria del nodo.
- Nodo de coordinación, este nodo solo puede manejar solicitudes de búsqueda y distribuir la indexación masiva. Son balanceadores de carga inteligentes porque obtienen la información del estado de los clústeres y tiene información sobre la tabla de enrutamiento de *shards* y los mapas de índices para saber a dónde debe enviar la solicitud.

3.1.3 Índice

A su vez, los nodos están formados por índices, que son una colección de documentos con características parecidas. Se suelen crear índices dependiendo de los tipos de documentos que almacenen, es decir, teniendo en cuenta el tipo. En los índices se realizan las operaciones de indexación, búsqueda, actualización y eliminación de los documentos que contiene.
Para poder representar los tipos de documentos, existía una categoría dentro de los índices llamada tipo de asignación, cada índice puede tener sus propios campos, pero los tipos de asignación deben tener la misma definición para mantener un número bajo de índices en los resultados de las consultas de búsquedas.

Para poder comprender mejor la plataforma Elastic, comparamos sus componentes con los de las BBDD SQL, un índice es similar a una base de datos y un tipo es similar a una tabla, pero las tablas son independientes entre sí y los tipos no son independientes entre sí porque deben seguir la misma definición, por esta razón, los tipos de asignación se acabaron eliminando de Elasticsearch, aunque fuera útil en las búsquedas.

3.1.4 Índice invertido

Dentro de los índices, la plataforma utiliza el llamado índice invertido que asigna términos a los documentos, para así agilizar la búsqueda de los documentos mediante los términos. Si se utiliza un índice directo, el funcionamiento sería recolectar los términos relacionados con un documento específico. Por consiguiente, a este concepto, para la búsqueda en Elastic se utilizarán frecuencias de términos, pero este punto se explicará más adelante.

3.1.5 Organización de datos: Fragmentación o réplicas

Como los índices pueden almacenar grandes cantidades de documentos, los hardware de los nodos pueden saturarse y por eso aparecen los conceptos de fragmentos y réplicas.
Los fragmentos son subdivisiones de los índices, funcionales e independientes que se pueden incorporar en cualquier nodo del clúster.

Existen dos métodos para organizar la información entre los nodos, siguiendo una estructuración distribuida, se puede separar mediante *Sharding* o fragmentación, que utiliza un mecanismo de tablas hash distribuidas (DHT) (Distributed Hash Table) (16), los datos se almacenan en varios servidores por subconjuntos.

Por otro lado, existe la réplica, son copias de los datos distribuidas por los servidores, por lo que la misma información puede encontrarse en varios sitios; las réplicas se pueden realizar mediante maestro-esclavo donde un servidor gestiona la escritura de la copia y los esclavos se sincronizan para las lecturas y, por otro lado, *peer-to-peer* donde cualquier nodo puede escribir y ellos se organizan para sincronizar las copias.

Para poder dividir los índices, se sigue el método de fragmentación explicado anteriormente, que además de dividir, también permite distribuir operaciones entre fragmentos y esto aumenta el rendimiento. Elasticsearch permite hacer una o más copias de los fragmentos en caso de que alguno de estos o que los nodos den algún error; a estas copias se las conoce como réplicas, aportan alta disponibilidad y escalabilidad horizontal. Es importante saber que las réplicas no pueden estar en el mismo nodo que el fragmento primario del que se copió.

3.1.6 Primera configuración

Para el correcto funcionamiento de Elastic, en primer lugar, existe una configuración de los archivos como ocurre en Logstash, por un lado, hay que configurar Elasticsearch, incluyendo los ajustes JVM (Java Virtual Machine) (17), configuraciones seguras y el registro.

Dentro de la configuración de Elasticsearch tenemos, la de la ruta que indican dónde se archivaran los datos; el nombre del grupo o clúster que por defecto es elasticsearch, pero debe ser cambiado según el propósito del clúster porque si no los nodos pueden confundir su ruta y el nombre del nodo; la configuración del host de red.

La configuración de descubrimiento es una de las más importantes porque sirve para que el nodo maestro conozca el estado y la cantidad de nodos que dispone el servidor, esto se consigue a través de un proceso de descubrimiento. Existe una matriz de hosts donde se almacenan las direcciones de los nodos. Otras configuraciones a tener en cuenta son: el tamaño de la pila, camino de descarga del montón, registros y el directorio temporal.

Elasticsearch debe ejecutarse en un único servidor y así emplear el resto como recursos, dependiendo del modo de trabajo, de desarrollo y de producción, los nodos podrán ser iniciados y ejecutados o no. Elasticsearch utiliza un directorio por defecto para almacenar sus índices. También al iniciarse la plataforma, se buscan todos los nodos que pertenecen al mismo host y en caso de que ocurra algún error, existen unos códigos de error ya registrados, ver Tabla 3-1.

Tabla 3-1: Registro de errores con sus códigos correspondientes

Nombre del error	Código del error
Error interno de JVM	128
Error de memoria insuficiente	127
Error de desbordamiento de pila	126
Error de máquina virtual desconocido	125
Error grave de E/S (Entrada / Salida)	124
Error fatal desconocido	1

3.1.7 Arquitectura interna

Cada componente de la plataforma tiene una API creada para la modificación de dicho componente, las funciones generales que comparten todos ellos son, la creación de índices, nodos, clúster, su modificación o eliminación. Además, para los documentos está implementada la funcionalidad de re-indexación; para los índices, la funcionalidad para generar el listado de índices creados, abrir y cerrar índices, gestión de mapeo y de estado, etc. Todas estas APIs realizan las operaciones a través del lenguaje JSON sobre HTTP.
Para comprender su analogía en la Tabla 3-2 se hace una comparación con BBDD MySQL.

Tabla 3-2: Analogía Elasticsearch comparada con una BBDD relacional

Elasticsearch	MySQL
Elasticsearch	Base de datos relacional
Índices	Base de datos
Tipos	Tablas
Documentos	Filas
Campos	Columnas

3.1.8 Mapeo

El mapeo es el proceso de definición de cómo se almacenan e indexan los documentos y los campos que contienen. Este es uno de los puntos fuertes de Elasticsearch al tratarse de una plataforma de almacenaje y búsqueda de datos.
En la especificación de mapeo se incluye, la especificación de los metadatos[3] y los campos que contendrán los documentos, los índices y los tipos de datos que serán estos campos.
Algunos de los tipos de mapeos actuales son la creación de índices por tipo de documento o la inserción del campo de tipo personalizado.

Una de las características importantes es que para indexar un documento no tiene que crear primero el índice y definir un tipo de mapeo, con indexar un documento sirve, automáticamente se asignarán un índice, tipo y campos; a esta característica se le llama mapeo dinámico.

3.1.9 Analizador

El análisis es el proceso de conversión del texto en *tokens* para agregarlos al índice invertido y así realizar la búsqueda. Para conseguir el resultado de *tokens*, las palabras deben pasar por un proceso donde se le aplican diferentes analizadores, pueden incluirse todos los analizadores necesarios, no existe un límite y todos ellos serán aplicados en orden.
Este proceso sigue unos pasos predefinidos:

1. Se convierte la frase en *tokens*. Esto se consigue mediante un filtro de caracteres que recibe el texto original y lo transforma a un flujo de caracteres. El

[3] Metadatos son un conjunto de datos que describen el contenido informativo de un recurso, archivo u otra información. Es decir, es información que describe otros datos.

"tokenizador" recibe un texto completo y lo divide en términos individuales generando un flujo de estos.
2. Se pasa a minúsculas cada *token*. Este paso se consigue con el filtro de *token*.
3. Se eliminan las palabras frecuentes como artículos. También conseguido mediante un filtro de *token*.
4. Se reducen los *tokens* a sus palabras.
5. Finalmente, las palabras que han quedado se agregan al índice invertido.

Existen diferentes tipos de analizadores que son enumerados en la Tabla 3-3:

Tabla 3-3: Tipos de analizadores

Tipo de analizador	Definición
Estándar	Divide el texto en *tokens*, elimina los signos de puntuación, los pasa a minúsculas.
Simple	Divide el texto en *tokens* cada vez que encuentra un carácter que no es una letra. Pasa a minúsculas los *tokens*.
De espacios en blanco	Divide el texto en *tokens* cada vez que encuentra un espacio en blanco.
De parada	Como el analizador simple, pero admite la eliminación de palabras de parada.
De palabras clave	Acepta cualquier texto y genera un *token* de este.
De patrones	Divide el texto mediante una expresión regular.
De lenguaje	Proporciona muchos analizadores de idiomas específicos.
De huellas dactilares	Crea una huella dactilar que puede usarse para detectar duplicados.
Personalizados	Se puede crear un analizar acorde a las necesidades.

3.1.10 Descubrimiento y formación de clúster

Se encarga de descubrir nodos, elegir uno que actúe como maestro, formar un clúster e informar de su estado cada vez que cambie. Su función sigue un procedimiento:

1. En primer lugar, se ejecuta el descubrimiento donde los nodos se encuentran entre sí. En el caso de añadir nuevos nodos a un clúster ya existente, el último clúster conocido aporta una lista de direcciones de todos los nodos y además las direcciones de los nodos aptos para ser maestro.
2. Mediante un mecanismo de votación basado en quórum se toman decisiones. El quórum es un subconjunto de los nodos elegibles para ser maestro en el clúster y las decisiones que se toman son las respuestas de los nodos que forman el quórum. Este trabajo en grupo de los nodos determina la decisión de qué nodo será maestro y los cambios del estado del clúster.
3. Se describen los conceptos de configuraciones de votación al unirse o salirse los nodos del clúster, la configuración se forma por el mismo conjunto quórum, esto

sirve para mantener un nivel óptimo de tolerancia de fallos. Para la elección de un nuevo nodo maestro o confirmar el estado del clúster se tienen en cuenta las respuestas del conjunto de nodos aptos para maestro. La decisión se toma después de que más de la mitad de los nodos en la configuración de votación haya votado.

4. Arranca el clúster.
5. Agregar y eliminar nodos aptos para maestro.
6. Publicar el estado del clúster es el modo en el que el nodo maestro actualiza la información del clúster en los demás nodos de este.
7. Detección de fallos del clúster, existen casos donde se consideran fallos, un caso es la detección de que un nodo ha sido desconectado para eliminar su información del clúster y otro caso sería al detectar que el nodo maestro ha sido desconectado donde se reinicia la fase de descubrimiento.

Para entender mejor el concepto, en la Figura 3-1 se pone como ejemplo un clúster que contiene tres nodos y los tres son aptos para maestro, por esto todos tienen su información dedicada a nodo maestro. Los tres nodos formarían el conjunto quórum y recibiendo sus respuestas se decidiría qué nodo debe ser el maestro. En el caso de que se elimine un nodo como pasa en la imagen, el quórum pasaría a estar formado por dos nodos.

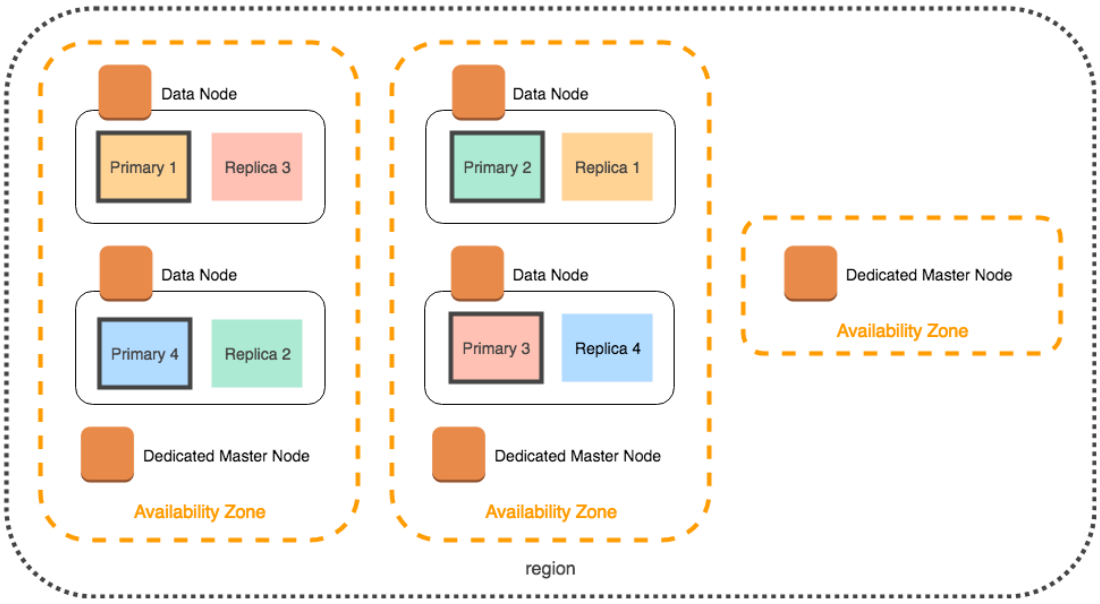

Figura 3-1: Esquema de la formación de un clúster por sus nodos (18)

3.2 Funcionamiento

Elasticsearch se compone de dos capas, un sistema distribuido encargado de la coordinación de nodos de un clúster y mantenimiento de sus datos y un motor de búsqueda con las funciones de indexación y búsqueda de documentos.

3.2.1 Primera capa: coordinación de nodos y mantenimiento de datos

En la primera capa, se arranca Elasticsearch y en este paso se crea un nodo que mediante el módulo "discovery" busca un nodo maestro con el mismo nombre del clúster, si lo encuentra se une a él y si no, lo crea y se asigna el nodo como maestro.
Para encontrar los nodos maestros, el módulo utiliza *Multicast* o *Unicast*. Con el primero, se manda un mensaje a todos los nodos de los grupos de máquinas y con el segundo se envía el mensaje a cada máquina. Por el contrario, mediante *Multicast* se pueden incorporar nodos a un clúster por error, y mediante *Unicast* el nodo debe conocer algún nodo del clúster para poder unirse a este por lo que no se puede dar ese caso de error.

Los nodos de datos se dividen en fragmentos que pueden ser primarios con funciones de lectura y escritura o de réplica con la única función de lectura. Estos fragmentos se reparten de forma automática entre los nodos del clúster tanto al insertar un nodo nuevo como al eliminar otro.
Al insertar los documentos, estos se distribuyen uniformemente entre todos los *shards*. En caso de eliminar un documento, se genera la petición, el nodo determina en que fragmento primario se encuentra el documento y se borra, a continuación, se propaga la operación a las réplicas.

3.2.2 Segunda capa: indexación y búsqueda de documentos

En la capa del motor de búsqueda, al llegar los datos desde Logstash, estos se deben almacenar en el índice especificado en una lista de direcciones o en el indicado mediante el mapeo. En primer lugar, el cliente genera una petición a un nodo, este determina en qué fragmento primario debe indexarse y una vez hecho este paso, el documento en el fragmento primario se propaga a las réplicas.

Pero si se realiza una consulta para recuperar información, el cliente realiza una petición a un nodo llamado coordinador, este determina en qué fragmento se encuentra el documento, envía la solicitud a los nodos de datos y estos realizan la solicitud de forma local, que consiste en realizar la consulta en cada fragmento, pasando por todos ellos mediante el método *round-robin* (19) y finalmente, devuelven los resultados al nodo coordinador que los reduce en un único conjunto de resultados.

Existen funcionalidades importantes a la hora de realizar búsquedas, una vez que se han recopilado los documentos en respuesta a la consulta, esta recopilación es binaria, solo se eligen los documentos con resultado 1 que quiere decir que contienen términos de la consulta y se deshacen de los documentos con valor 0.
Los documentos elegidos se ordenan mediante la relevancia de los documentos, la relevancia es un valor numérico calculada sobre un conjunto de resultados basada en el valor de un término, la interpretación de una búsqueda, aunque tenga errores ortográficos, interpretar búsquedas en diferentes idiomas y la predicción de alternativas.

El *score* o frecuencia es un valor decimal que determina el orden sobre el resultado de una búsqueda. El algoritmo que utiliza Elasticsearch para calcular la relevancia de un documento hace uso de la fórmula de relevancia de Lucene, calculando el peso de un término en cada documento.
El algoritmo combina varios cálculos para conseguir el resultado deseado, a este algoritmo se le denomina como *practial scoring function:*

$$score(q,d) = qN(q) \times coord(q,d) \times \sum (tf(t\ en\ d), idf(t)^2, t.getBoost(), N(t,d))(t\ en\ q)$$

En la ecuación se incluye la combinación de cálculos que se van a explicar por partes para su comprensión.
En primer lugar, hay que conocer la puntuación de relevancia (score (q, d)) de una consulta (q) en un documento (d). Cuando en las fórmulas se refiera a la variable t, equivale al término que se busca y la variable d al documento donde se realiza la búsqueda.

qN(q) es el factor de normalización de la consulta, esto se refiere a la aplicación de filtros para sacar el término deseado, es decir, normalizar una consulta para que los resultados puedan compararse con otros. Podemos ver su formulación en la siguiente ecuación.

$$qN(q) = {}^{1}/_{\sqrt{\sum(idf(t)^2)}}$$

coord(q, d) se refiere al factor de coordinación de las consultas, y esto se usa para recompensar los documentos que contienen un porcentaje más alto de los términos de la consulta, cuantos más términos aparezcan en el documento, mayor será la posibilidad de que el documento coincida con la consulta. Este factor multiplica la puntuación del término por el número de términos coincidentes en el documento y lo divide por el total de términos en la consulta.

$$coord(q,d) = {}^{weight(t) \times num(t)}/_{num(t\ en\ q)}$$

La suma de los resultados del peso de cada término de la consulta en cada documento nos lleva al concepto de *term frequency (tf)* e *inverse document frequency* (*idf*) que calculan el número de veces que aparece el término en un documento, cuantas más veces aparezca, más relevante será el documento y mayor será la relevancia.

$$tf(t\ en\ d) = \sqrt{frecuencia}$$

idf(t), es la frecuencia inversa del término t en el documento, significa cuántas veces aparece el término con respecto a todos los documentos, es decir, con respecto a todo el índice. maxDocs es el número de documentos en el índice y docFreq es el número de documentos que contienen el término buscado.
Cuantas más veces aparezca en más documentos, menor será la relevancia.

$$idf(t) = 1 + \ln {}^{maxDocs}/_{docFrec + 1}$$

t.getBoost() es un valor de prioridad que se ha aplicado a la consulta, esto quiere decir que dependiendo de lo que busca el cliente, se le puede dar prioridad a los documentos indexados recientemente que contengan los términos frente a los más antiguos. La función

getBoost() devuelve cualquier valor de prioridad que haya sido aplicada al término o a la consulta.

N(t, d) es la norma de longitud de campo, cuanto más corto sea el campo, mayor será el peso. Esto se debe a que, si un término aparece en un campo corto como un título, es más probable que el contenido de ese campo sea sobre el término en sí que si aparece en un campo de cuerpo más grande.

$$N(t,d) = {}^{1}/_{\sqrt{num(t)}}$$

3.3 Calidad de rendimiento

Como ya se ha hablado a lo largo del capítulo, Elasticsearch aporta un alto rendimiento al servicio que lo utilice, las búsquedas y almacenamientos de datos son óptimos debido al método de indexación que utiliza la plataforma. Pero existen factores que pueden afectar al rendimiento de un servicio que utilice Elasticsearch y uno de los factores son los *shard*, encargados de distribuir los datos en el clúster.

La velocidad a la que Elasticsearch puede mover *shards* depende del tamaño y la cantidad de estos, otros factores a tener en cuenta es el rendimiento de la red y el disco.
La memoria almacena información de mapeos y estados de cada índice en el estado del clúster, en la memoria para ofrecer un rápido acceso. Por esta razón, tener mucha cantidad de *shards* e índices en un clúster puede hacer que el estado del clúster sea grande y se pueden encontrar actualizaciones lentas por su procesamiento en un único hilo.

Los *shards* pequeños aumentan la sobrecarga, la idea es mantener un tamaño promedio, lo ideal es mantener una cantidad de *shards* por nodo por debajo de 20 por GB de heap configurado.
Si se consultan múltiples *shards* pequeños, cada hilo por *shard* será más rápido, pero habrá tareas que se almacenen en cola; pero no quiere decir que sea más rápido que consultar en menos *shards* de mayor tamaño cada uno.

Como se ha observado a lo largo del capítulo la flexibilidad que ofrece Elasticsearch es una ventaja que los clientes deben conocer, cuando tienen un clúster grande y lo quieren ampliar deben saber que a ese crecimiento le precede una planificación y diseño para conseguir que no se den errores en el proceso.

Para conocer la cantidad exacta de *shards* que se deben configurar hay que tener en cuenta otras variables como el tipo de hardware, el tamaño y complejidad de los documentos, la forma de indexar y analizar los documentos, los tipos de consultas que soportará el servicio, etc. Si se empieza por la capacidad de un *shard*, conociendo el total de datos que se quiere indexar, más algunos adicionales para el crecimiento futuro y se divide por la capacidad de un *shard* entonces se podrá estimar la cantidad de *shards* necesarios. En la Figura 3-2 se considera un ejemplo de nodos con sus *shards*, este ejemplo sirve para ver que además de la cantidad de *shards* a planificar, también se debe contar con sus réplicas porque estas ocupan el mismo espacio que los *shards* primarios.

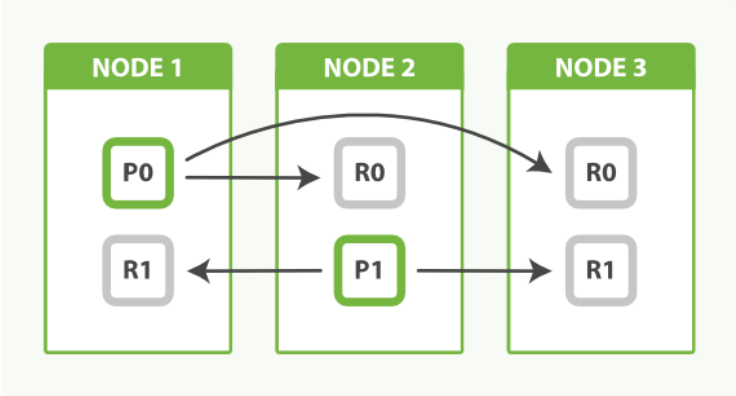

Figura 3-2: División de *shards* entre los nodos de un clúster (20)

3.4 Casos de uso

Algunos ejemplos de casos de usos de Elasticsearch:

- En una tienda web se puede utilizar para almacenar el catálogo y permitir que los clientes hagan consultas de los productos e inventario y ofrecerle sugerencias mediante autocompletado.
- Se pueden recopilar datos de transacciones, analizarlos y extraerlos para buscar estadísticas o resúmenes.

3.5 Alternativa Apache Solr

Apache Solr es un motor de búsqueda basado en Lucene, fue la primera versión de lo que ahora conocemos como Elasticsearch, aunque hoy en día han evolucionado para dar diferentes ofertas.

Por un lado, sabemos que Elasticsearch tiene una implementación más sencilla y es más flexible al cambiar datos que Solr, aunque esta última está respaldada por la comunidad de Apache.

Elastic aporta su documentación a través de Github (21) mientras Solr la aporta mediante Atlassian Confluence (22).

Solr está orientado a la búsqueda como motor de búsqueda mientras Elasticsearch a través de la API ELK es capaz de analizar los datos también.

Su uso variará dependiendo de los requisitos que tenga el cliente ya que ambos son dos motores de búsqueda y BBDD no relacionales potentes.

4 Kibana

4.1 Definición, conceptos y terminología

Kibana es una plataforma de análisis y visualización diseñada para trabajar con Elasticsearch. Su función es permitir al usuario ver e interactuar con los datos almacenados en los índices de Elasticsearch. Puede visualizar los datos en tablas y mapas en tiempo real.

La necesidad de una plataforma como Kibana se debe a facilitar al usuario la interacción con el motor de búsqueda, las solicitudes que se hacen son las mismas, mediante JSON mediante request HTTP, la diferencia es la interfaz creada para que los usuarios puedan realizar búsqueda, filtrar los resultados y organizarlos en diferentes formatos que se irán explicando brevemente a lo largo de este apartado.

4.1.1 Discover

Es una función de Kibana que le permite explorar los datos ya que tiene acceso a cada documento de cada índice que coincida con el patrón de índice seleccionado.
Puede enviar consultas de búsqueda, filtrar los resultados y ver los datos de documentos. También puede ver la cantidad de documentos que coinciden con la consulta y sacar estadísticas.

En el caso de querer actualizar los datos por medidas de tiempo, existe un filtro de tiempo que puede establecer o crear un intervalo de tiempo. Algunas maneras de filtrar este selector de tiempo son:

- Rápida, se puede seleccionar un tiempo entre las opciones que te ofrece el programa.
- Relativa, pide especificar un rango de tiempo teniendo como marca la hora actual.
- Absoluta, especifica el inicio y el fin de un intervalo de tiempo.
- Reciente, puede seleccionar cualquiera de los tiempos utilizados recientemente.

También puede realizar una búsqueda a través del filtrado por campo, donde se muestran solo los documentos que contienen un valor particular en un campo. Los filtros pueden ser en negativo o en positivo y pueden utilizarse para recuperar documentos con un valor específico o extraer los documentos con ese mismo campo filtrado. Como en la mayoría de los buscadores, los filtros pueden asignarse de forma fija y que se mantengan para las consultas que el cliente requiera.

4.1.2 Visualize

Es otra función que permite crear vistas de los datos en sus índices. Las visualizaciones son consultas de Elasticsearch de las cuales se pueden crear gráficos de líneas y las tendencias que el cliente necesite conocer.

Existen diferentes tipos de visualización:

- Cuadros básicos que incluyen gráficos de líneas, áreas y barras, gráfico circular.

- Datos, tabla de datos, número único por selección de datos, objetivos e indicadores que muestran el progreso de unos datos métricos a un objetivo fijo o en qué rango los resultados bajan.
- Mapas de coordenadas o de la región.

4.1.3 Tablero

Puede crear una colección de visualizaciones y búsquedas en un tablero y compartirlo. En la Figura 4-1 se muestra un ejemplo donde se pueden ver los distintos tipos de visualizaciones nombradas y cada uno de ellos muestra datos que en este caso el cliente necesita conocer o mantener actualizados para el correcto funcionamiento de su aplicación.

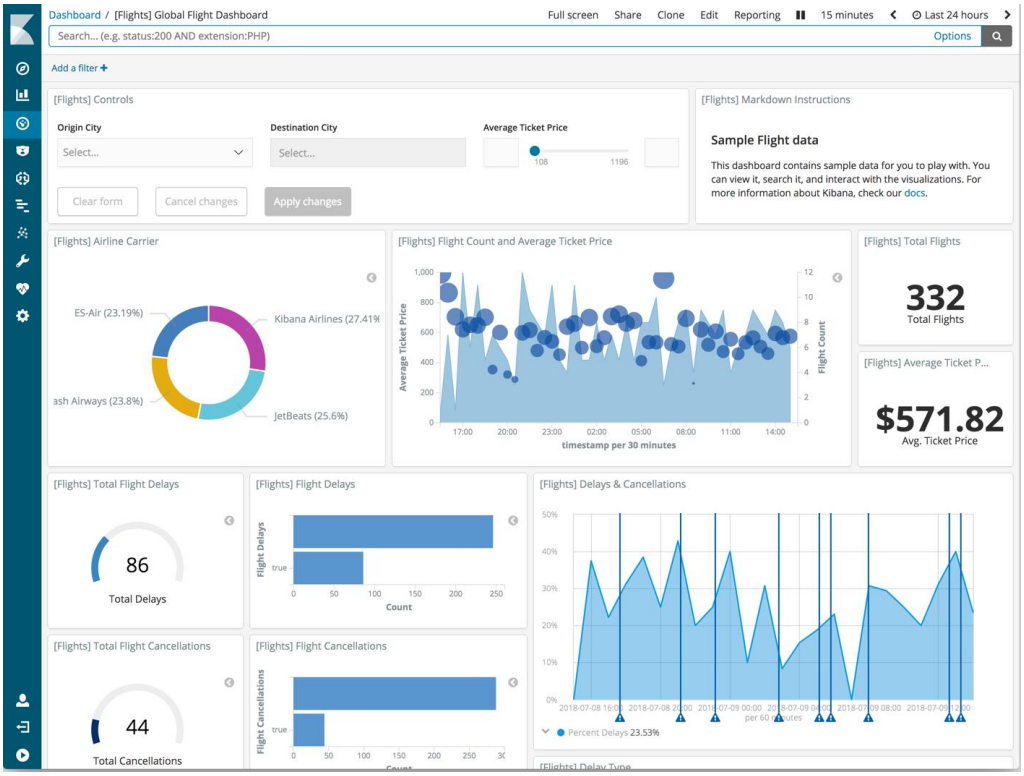

Figura 4-1: Tablero de ejemplo con una colección de visualizaciones y búsquedas en Kibana (23)

4.1.4 Lienzo

Para el uso de este pack se debe conocer la plataforma Canvas (24), tiene una función similar a la del tablero, la diferencia es que mediante esta aplicación se pueden generar visualizaciones más estéticas y creativas. La Figura 4-2 muestra un ejemplo para poder comparar las herramientas y apreciar las diferencias.

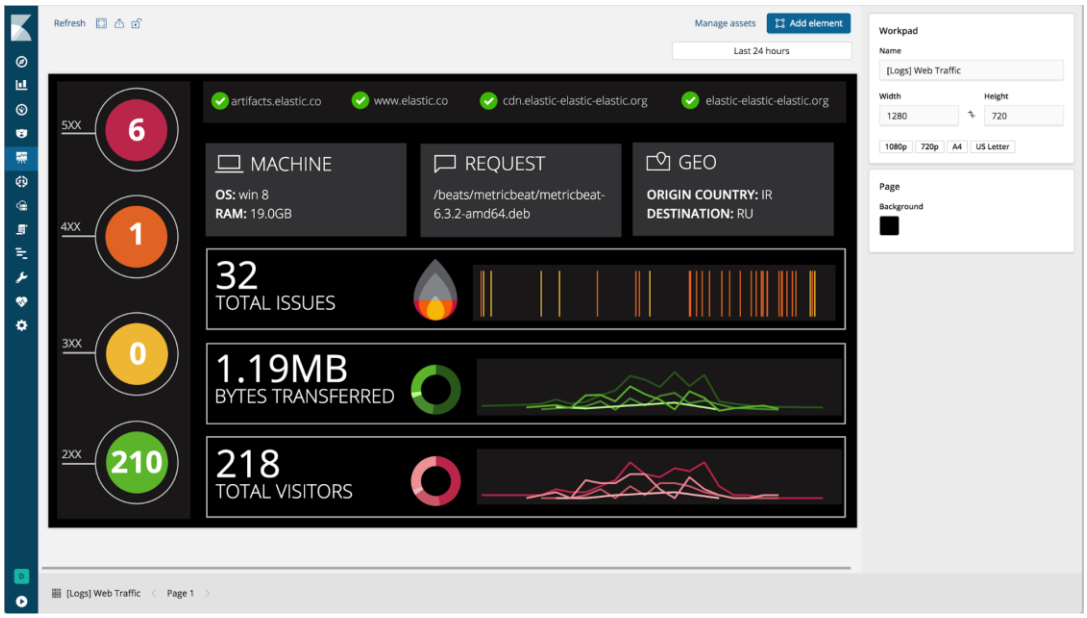

Figura 4-2: Ejemplo de visualización de Canvas en Kibana (25)

Existen otras aplicaciones para realizar análisis de los datos en diferentes modalidades, como son: los mapas, para analizar los datos de forma geográfica.

4.1.5 API de registros

El caso de uso más común se da con la interfaz de usuario de logs, sirve para consultar los registros de servidores, contenedores y servicios comunes. Es uno de los usos más comunes porque muestra los datos a través de una consola

4.1.6 API de infraestructura

Es una infraestructura interactiva para monitorizar e identificar problemas en tiempo real. Puede sacar métricas y registros de servidores o contenedores como en el caso anterior.
Para recopilar errores de rendimiento en las aplicaciones de Elasticsearch, se utiliza la aplicación llamada APM (Application Performance Monitoring) que es capaz de visualizar en qué puntos de la aplicación se generan cuellos de botella de rendimiento.

Los cuellos de botella se pueden dar por información clasificada cómo intervalos, transacciones o búsquedas, solicitudes, etc. y errores sobre registros o excepciones que se den durante todo el proceso de la ELK Stack.

Toda esta información se muestra en tablas o cuadros para poder compararlos y depurar las aplicaciones, por ejemplo, con los tiempos de respuesta, viendo en qué punto está la aplicación dedicando más tiempo.

Normalmente los cuellos de botella se dan en las consultas de bases de datos, solicitudes externas; todo este conocimiento puede ayudar a disminuir el tiempo dedicado a errores de depuración y retrasar tiempos de respuesta que más tarde puedan bloquear la aplicación.

4.2 Aprendizaje automático

Con la gran cantidad de datos que procesa Elasticsearch, se requiere un mayor esfuerzo por parte del cliente para identificar errores y anomalías mientras Elastic modela el comportamiento normal de los datos en tiempo real; identificar estos errores rápidamente ayuda a encontrar la causa y reducir los falsos positivos.

Existe una función en Kibana llamada aprendizaje automático donde el usuario puede entender los resultados y detectar anomalías gracias a que capta lo que se puede perder de forma automática.

Este aprendizaje automático es realmente un modelaje automático del comportamiento de los datos en tiempo real, tendencias, periodicidad, etc. Por ejemplo, puede detectar una bajada en las solicitudes de aplicaciones y una vez detectado, profundizar en el servidor con el problema. También puede identificar la actividad de la red o los usuarios y así encontrar atacantes antes de que causen daños; o encontrar solicitudes de compras abandonadas en un sitio de la página del comercio mediante la llegada de una notificación.

Otro caso de uso de esta función es el cuidado del rendimiento, detectando cuellos de botella y ralentizando tiempos de respuesta para el correcto funcionamiento de las aplicaciones.

Para aprovechar esta función, se debe describir los datos que se quieren analizar y las propiedades que puedan influir en ellos. El modelo considera lo que es normal y detecta lo que se desvía de la norma.

En la Figura 4-3 se puede ver un ejemplo de la ventana aprendizaje automático con unos datos que siguen una figura y cómo en ciertos momentos la figura sigue un diseño y la representación lineal de los datos otras, este sería un caso de anomalía que el cliente puede identificar gracias a la función de aprendizaje automático.

Figura 4-3: Ejemplo de una visualización de la ventana aprendizaje automático (26)

5 Uso de ELK en grandes compañías

Elasticsearch se ha convertido en una herramienta muy importante en las grandes compañías debido a la flexibilidad y mejoría en el procesamiento de grandes cantidades de datos.

Algunas grandes empresas que han utilizado esta plataforma son: Dell, Linkedin, Facebook, Telefónica, Just Eat, Accenture, GitHub y muchos otros.

5.1 Caso de uso: Dell

En el caso de Dell, su función es apoyar la búsqueda de comercio electrónico, es una empresa que estaba teniendo problemas al ofrecer sus productos a los clientes, su buscador no admitía multi-cliente y se quedaba obsoleto por lo que buscaron una alternativa entre Solr, Google Search Appliance y otros motores de búsqueda. Finalmente se decantaron por Elasticsearch debido a su fácil escalabilidad, relevancia en resultados y código abierto para ofrecer un resultado al cliente más personalizado.

Dell ha utilizado dos clústeres, uno dedicado a la búsqueda y otro para el análisis de la actividad del usuario, todo dentro de su página web. El dedicado a las búsquedas almacena todos los productos de la marca, software, soluciones de problemas, documentos de base de conocimiento, manuales, estado de stock, información de precios, etc. La parte de análisis se encarga de ofrecer a los clientes resultados de búsqueda influidos por otros usuarios y comprensión del uso de la plataforma de búsqueda y obtener análisis para mejorar la dinámica y acciones de la página.

5.2 Caso de uso: LinkedIn

Otra empresa que ha implementado Elasticsearch en su plataforma es Linkedin, debido a la gran cantidad de usuarios a los que da soporte y sobre todo al uso que estos le dan al programa, su principal problema era la cantidad de datos, servidores y entradas que soporta, la visualización y análisis de estos componentes cada día, el nivel de seguridad que tenía que ser capaz de soportar y el aumento de escalabilidad, a mayor número de logs, mayor cantidad de servidores, etc.
Al incorporar Elasticsearch en su plataforma, su arquitectura queda similar a la de la Figura 5-1 donde se ve que los usuarios interaccionan directamente con Kibana y este manda las peticiones al nodo maestro de Elasticsearch que consulta los datos de los nodos de datos, donde estos últimos han sido recolectados mediante la plataforma Kafka en defecto de Beats y traducida por Logstash.

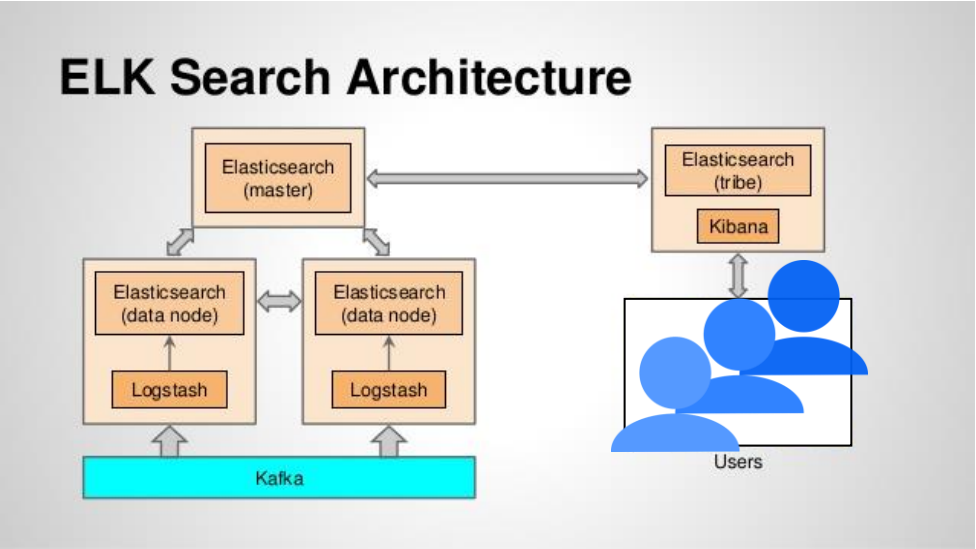

Figura 5-1: Arquitectura general de Elasticsearch implementado en Linkedin (27)

5.3 Caso de uso: Facebook

En el caso de Facebook, pasaron por otras plataformas antes de llegar a Elasticsearch, por ejemplo, consultaron la posibilidad de Google Search Appliance, pero encontraron algunas desventajas como que no podían acceder a la estructura de datos y se encontraron con una caja bloqueante por lo que pasaron a probar con Apache Solr, pero se encontraron con que tenían que crear herramientas que Elasticksearch le ofrecían y el lenguaje de las consultas debía ser en XML. Por estos ejemplos, finalmente se decantaron por el motor Elasticsearch, porque está basado en Lucene, pero aporta una interfaz simplificada, es fácil de instalar y hacer funcionar y por estar desarrollada en código abierto.

5.4 Caso de uso: Telefónica

Una de las empresas con más prestigio en nuestro país, Telefónica, también utiliza este servicio; con los servicios que ofrece y el consumo que tiene, la empresa visualizó un crecimiento de logs de datos y métricas. Por un lado, el desarrollo de infraestructuras internas está solventado, pero no tenían forma de extraer, unificar ni analizar los datos de sus sistemas en tiempo real.
Encontraron la solución con ELK y están trabajando en una plataforma de gestión de datos para mejorar la experiencia del cliente y de acceso en tiempo real al valor operativo y comercial.
Una de sus apuestas más fuertes son los servicios de video, comunicación móvil e internet y con su crecimiento vieron necesaria la implementación de una plataforma en Elasticsearch para obtener información como el consumo de los clientes y rendimiento del servicio. Toda esta información antes pasaba desapercibida, pero gracias a esta plataforma, pueden supervisar la proporción de visualización de forma filtrada, en directo, bajo demanda o por regiones.
Otro ejemplo, es la consulta de errores en los fragmentos de videos para determinar tasas de uso y así centrar a sus equipos en los trabajos, notificar con más detalle sobre los problemas que dan sus equipos. Además, en la detección de problemas, utilizan la característica de aprendizaje automático de ELK para modelar de forma automática el

comportamiento de las tendencias de datos y así ayudar en la detección de anomalías, analizando la causa raíz y reduciendo la cantidad de falsos positivos.

5.5 Caso de uso: Just Eat

En el caso de una plataforma de reparto como es Just Eat, el uso de Elastic ayuda en la satisfacción del cliente, porque el principal problema que le perjudica es la tasa de repartos y si los pedidos son incorrectos. Gracias al monitoreo continuo con Logstash y Kibana el equipo puede reaccionar rápido y aportar precisión en las entregas. También proporciona actualizaciones en tiempo real cada vez que un restaurante realiza un cambio.
Los principales objetivos en las búsquedas de la empresa son, estabilidad, tiempo de respuesta y precisión de los resultados, el equipo necesitaba una herramienta de búsqueda geoespacial y por esto Elasticsearch fue su apuesta a largo plazo.

5.6 Caso de uso: Accenture

En el caso de Accenture, el objetivo principal era aportar una búsqueda más precisa a sus empleados debido a que esta era una de las funciones que menos utilizaban para desarrollar proyectos porque su plataforma, en aquel momento, no era flexible y no aportaba la información precisa que requerían los empleados. Además, esto empeoraba a medida que la cantidad de documentos aumentaba por la cantidad de proyectos que desarrolla la consultora. Como es una empresa que se basa en la mejora y la actualización de plataformas, se decantó por Elasticsearch donde colocan los datos en un "índice sucio" y una vez los procesan y normalizan, estos pasan a un "índice limpio" donde se ejecutan las consultas.
El verdadero beneficio, es una búsqueda más rápida y relevante para así fomentar el uso de sus recursos y conocimientos para ofrecer un mejor servicio a los clientes.

5.7 Caso de uso: GitHub

Para GitHub el reto consistía en satisfacer las necesidades de búsqueda de sus usuarios a la vez que aportar información para mejorar el servicio al cliente. Al principio, utilizó Solr para las búsquedas, pero encontró que esta no podía escalar de forma efectiva y tenía una administración más compleja. Se encontraron en el punto donde fragmentar sus propios datos en Solr o pasarse a la plataforma de Elasticsearch ya que ofrece un reequilibrio automático de fragmentos para aumentar el rendimiento. También utiliza las consultas avanzadas y la monitorización de su infraestructura interna para buscar abusos y errores por parte de los usuarios. Ofrece un alto rendimiento ya que no fragmenta los documentos en varias partes y utiliza documentos enteros como única fragmentación para las búsquedas dentro de un solo repositorio, duplicando la velocidad de respuesta.

6 Conclusiones

Tras estudiar el nacimiento de la plataforma, la arquitectura interna, su funcionamiento y evaluar todo lo que puede aportar Elasticsearch, mediante sus funciones o conjunto de plataformas que la complementan, podemos sacar algunas conclusiones como:

- Al estar construida sobre Lucene que es una biblioteca de recuperación de información, proporciona opciones de búsqueda más potentes por su personalización en la formación de las BBDD.
- Permite una buena búsqueda a pesar de los errores ortográficos gracias a la funcionalidad de autocompletado. Aporta un alto rendimiento gracias al uso de documentos JSON estructurados e indexados mediante todos los campos que lo forman.
- Su velocidad para ejecutar consultas complejas y el almacenaje de las consultas más comunes como filtro es otro beneficio que los clientes apoyan.
- El uso de un sistema distribuido, capaz de ser escalado horizontalmente para así poder equilibrar la carga de trabajo entre los nodos del clúster. Los índices al dividirse en fragmentos y estos contener réplicas, el enrutamiento se realiza de forma automática siempre que se indexan nuevos documentos.

Todos estos beneficios son los que han conseguido que Elasticsearch sea una de las plataformas más utilizadas en las grandes empresas. Al ser una plataforma joven, la necesidad de mejorar y modernizar los recursos de las empresas le ha aportado ese crecimiento que Solr no ha conseguido obtener por su aparición más temprana y debido a esto su poca flexibilidad.

Las plataformas nacen con las necesidades de las personas y este es un claro ejemplo donde se puede ver que Elasticsearch llegó con la necesidad de las empresas de aportar una búsqueda más personalizada y con un soporte de una cantidad de datos que aumenta de forma constante.

Sus funciones de visualización también se deben a las necesidades de los clientes de monitorizar los datos y buscar siempre la mejora, aportando unos escritorios con analizadores más desarrollados y garantizando el estudio de los datos que indexa la plataforma, beneficiando así a los clientes con el estudio de errores en sus aplicaciones o aportando datos de rendimiento para buscar nuevas funciones y crecimiento a sus servicios.

Además, gracias a la implementación de aplicaciones con datos de distintos tipos, Elasticsearch ha conseguido aportar a sus clientes diferentes Beats dependiendo de los tipos de datos que se quieren incorporar en sus BBDD. Todo el crecimiento que realiza la plataforma de Elasticsearch y las funciones que se incorporan de manera continua se debe a la necesidad de estar actualizado con las necesidades de las empresas y estas a su vez de sus clientes.

Para terminar, se ha analizado la plataforma desde su nacimiento, viendo las notables diferencias con las BBDD convencionales y se puede considerar que su aparición ha sido necesaria; gracias a esta plataforma las empresas y las aplicaciones empezaron a implementarla porque un servicio personalizado siempre aporta más beneficios que

pérdidas y a la hora de crear nuevas aplicaciones los objetivos seguirán siendo facilitar el trabajo a la persona emprendedora y dar mejor soporte a los clientes para su satisfacción.

Referencias

1. Bases de datos NoSQL: Guía definitiva. [En línea] PandoraFMS, 20 de Abril de 2017. https://pandorafms.com/blog/es/bases-de-datos-nosql/.[En línea]
2. https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf.
3. Apache Cassandra. [En línea] http://cassandra.apache.org/.
4. Bigtable. [En línea] Wikipedia, 14 de Septiembre de 2017. https://es.wikipedia.org/wiki/BigTable.
5. Neo4j. [En línea] https://neo4j.com/.
6. InfoGrid. [En línea] SlideShares. https://es.slideshare.net/infogrid/the-infogrid-graph-database.
7. Logstash canalización básica. [En línea] https://www.elastic.co/guide/en/logstash/current/first-event.html.
8. Filebeat Beats. [En línea] https://www.elastic.co/es/products/beats/filebeat.
9. Metricbeat Beats. [En línea] https://www.elastic.co/es/products/beats/metricbeat.
10. Packetbeat Beats. [En línea] https://www.elastic.co/es/products/beats/packetbeat.
11. Winlogbeat Beats. [En línea] https://www.elastic.co/es/products/beats/winlogbeat.
12. Auditbeat Beats. [En línea] https://www.elastic.co/es/products/beats/auditbeat.
13. Heartbeat Beats. [En línea] https://www.elastic.co/es/products/beats/heartbeat.
14. Functionbeat Beats. [En línea] https://www.elastic.co/es/products/beats/functionbeat.
15. Apache Lucene. [En línea] Lucene. https://lucene.apache.org/.
16. Tabla de hash distribuida Wikipedia. [En línea] https://es.wikipedia.org/wiki/Tabla_de_hash_distribuida
17. Máquina Virtual Java. [En línea] Wikipedia, 4 de Junio de 2019. https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java.
18. AWS. [En línea] https://docs.aws.amazon.com/es_es/elasticsearch-service/latest/developerguide/es-managedomains.html.
19. Planificación Round-robin. [En línea] Wikipedia, 16 de Octubre de 2018. https://es.wikipedia.org/wiki/Planificaci%C3%B3n_Round-robin.
20. Réplica Shards. [En línea] Elastic. https://www.elastic.co/guide/en/elasticsearch/guide/2.x/replica-shards.html.
21. Github. [En línea] Wikipedia, 28 de Mayo de 2019. https://es.wikipedia.org/wiki/GitHub.
22. Confluence. [En línea] Atlassian. https://es.atlassian.com/software/confluence.
23. Panel de control de Kibana. [En línea] https://www.elastic.co/guide/en/kibana/current/dashboard.html.
24. Canvas HTML. [En línea] Wikipedia. https://es.wikipedia.org/wiki/Canvas_(HTML). (HyperText Markup Language)
25. Canvas. [En línea] ElasticDocs. https://www.elastic.co/guide/en/kibana/current/canvas.html#canvas.
26. Elastic. [En línea] https://www.elastic.co/es/products/kibana.
27. Tin Le, Senior Site Reliability Engineer at LinkedIn. LinkedIn SlideShare. [En línea] 2 de Mayo de 2015. https://es.slideshare.net/TinLe1/elk-atlinked-in.